24. Geometric Algorithms

Properties of Line Segments, Intersection of Line Segments, Convex Hull, Closest Point Pair [Ottman/Widmayer, Kap. 8.2,8.3,8.8.2, Cormen et al, Kap. 33]

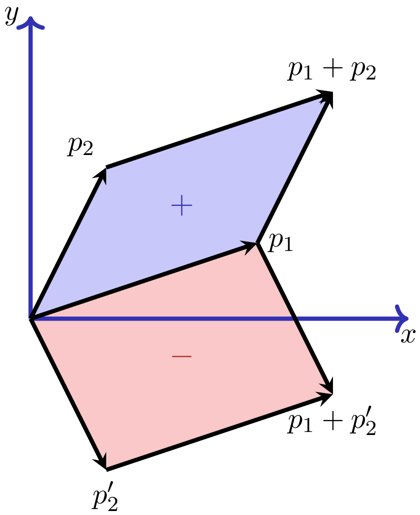
24.1 Properties of Line Segments

Properties of line segments.

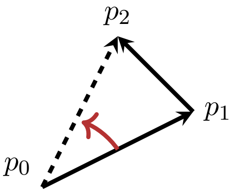
Cross-Product of two vectors $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$ in the plane

$$p_1 \times p_2 = \det \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix} = x_1 y_2 - x_2 y_1$$

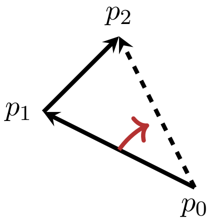
Signed area of the parallelogram



Turning direction

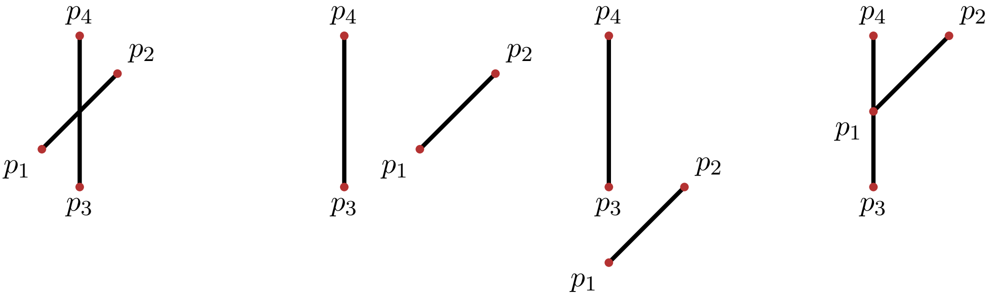


nach links:
$(p_1 - p_0) \times (p_2 - p_0) > 0$



nach rechts:
$(p_1 - p_0) \times (p_2 - p_0) < 0$

Intersection of two line segments



Intersection: p_1 and p_2 opposite w.r.t $\overline{p_3p_4}$ and p_3, p_4 opposite w.r.t. $\overline{p_1p_2}$

No intersection: p_1 and p_2 on the same side of $\overline{p_3p_4}$

No intersection: p_3 and p_4 on the same side of $\overline{p_1p_2}$

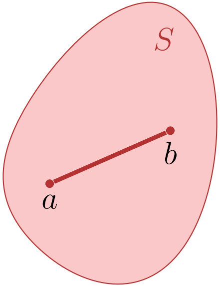
Intersection: p_1 on $\overline{p_3p_4}$

24.2 Convex Hull

Convex Hull

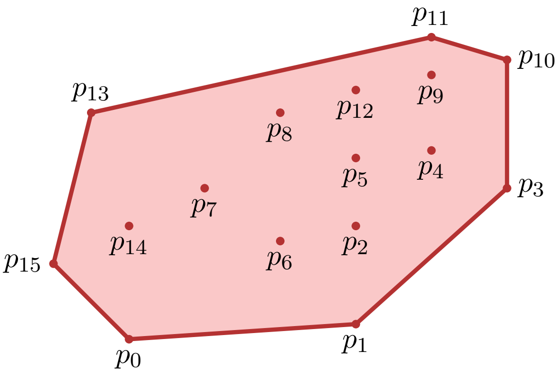
Subset S of a real vector space is called **convex**, if for all $a, b \in S$ and all $\lambda \in [0, 1]$:

$$\lambda a + (1 - \lambda) b \in S$$



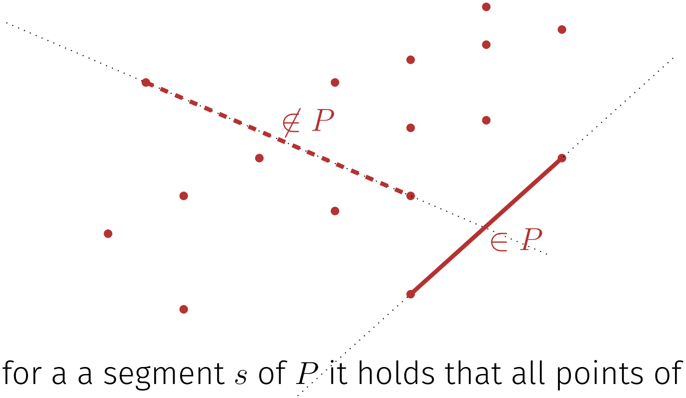
Convex Hull

Convex Hull $H(Q)$ of a set Q of points: smallest convex polygon P such that each point of Q is on P or in the interior of P.



Convex Hull

Identify segments of P

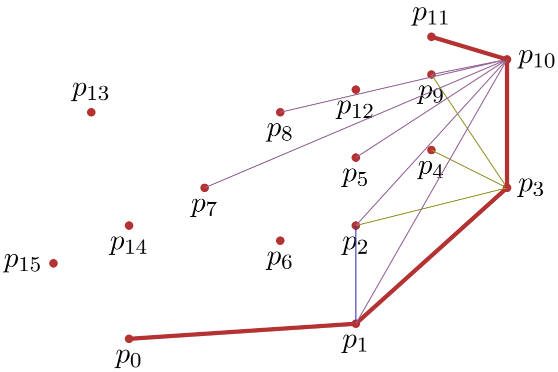


Observation: for a a segment s of P it holds that all points of Q not on the line through s are either on the left or on the right of s.

Jarvis Marsch / Gift Wrapping algorithm

1. Start with an extremal point (e.g. lowest point) $p = p_0$
2. Search point q, such that $\overline{pq}$ is a line to the right of all other points (or other points are on this line closer to p.
3. Continue with $p \leftarrow q$ at (2) until $p = p_0$.

Illustration Jarvis

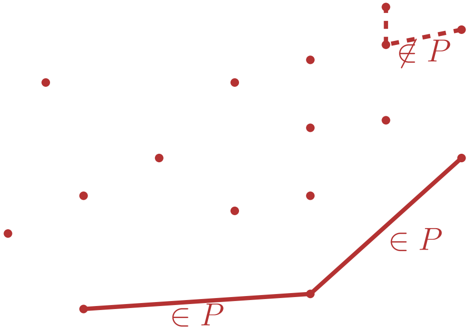


Analysis Gift-Wrapping

- Let h be the number of corner points of the convex hull.
- Runtime of the algorithm $\mathcal{O}(h \cdot n)$.

Convex Hull

Identify segments of P

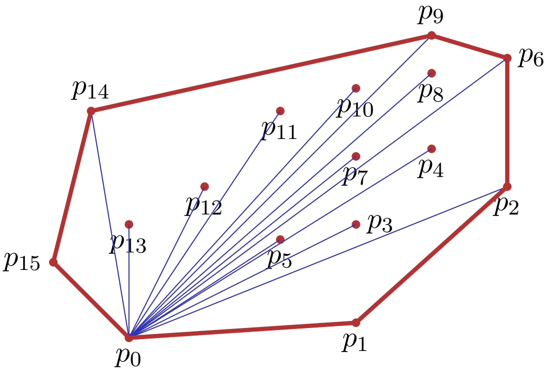


Observation: if the points of the polygon are ordered anti-clockwise then subsequent segments of the polygon P only make left turns.

Algorithm Graham-Scan

Input: Set of points Q
Output: Stack S of points of the convex hull of Q
p_0: point with minimal y coordinate (if required, additionally minimal x-) coordinate
$(p_1, \ldots, p_m)$ remaining points sorted by polar angle counter-clockwise in relation to p_0; if points with same polar angle available, discard all except the one with maximal distance from p_0
$S \leftarrow \emptyset$
if $m < 2$ **then return** S
Push(S, p_0); Push(S, p_1); Push(S, p_2)
for $i \leftarrow 3$ **to** m **do**
  while Winkel (NextToTop(S), Top(S), p_i) nicht nach links gerichtet **do**
    Pop(S);
  Push(S, p_i)
return S

Illustration Graham-Scan



Stack:

p_{15}

p_{14}

p_9

p_6

p_2

p_1

p_0

Analysis

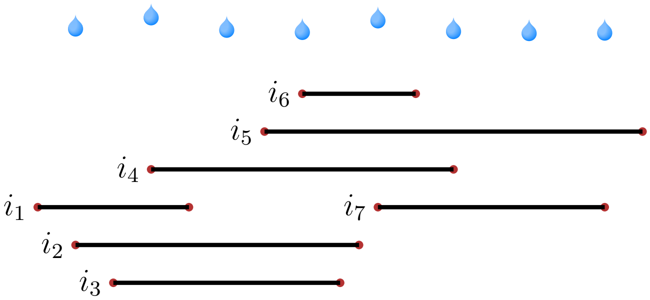
Runtime of the algorithm Graham-Scan

- Sorting $\mathcal{O}(n \log n)$
- n Iterations of the for-loop
- Amortized analysis of the multipop on a stack: amortized constant runtime of mulitpop, same here: amortized constant runtime of the While-loop.

Overal $\mathcal{O}(n \log n)$

24.3 Intersection of Line Segments

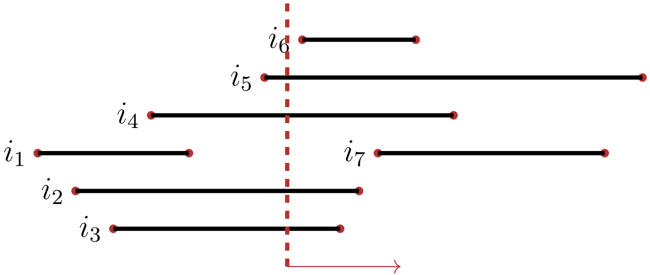
Preparation: Overlapping Intervals



Questions:

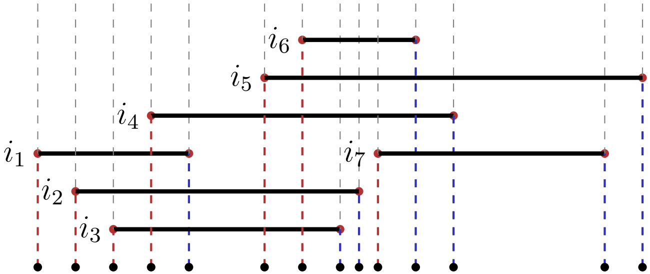
- How many intervals overlap maximally?
- Which intervals (don't) get wet?
- Which intervals are directly on top of each other?

Preparation: Overlapping Intervals



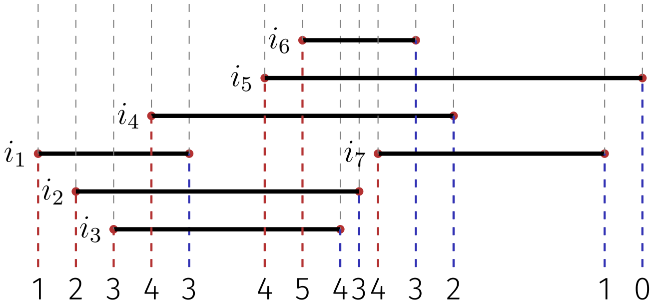
Idea of a sweep line: vertical line, moving in x-direction, observes the geometric objects.

Preparation: Overlapping Intervals



Event list: list of points where the state observed by the sweepline changes.

Preparation: Overlapping Intervals

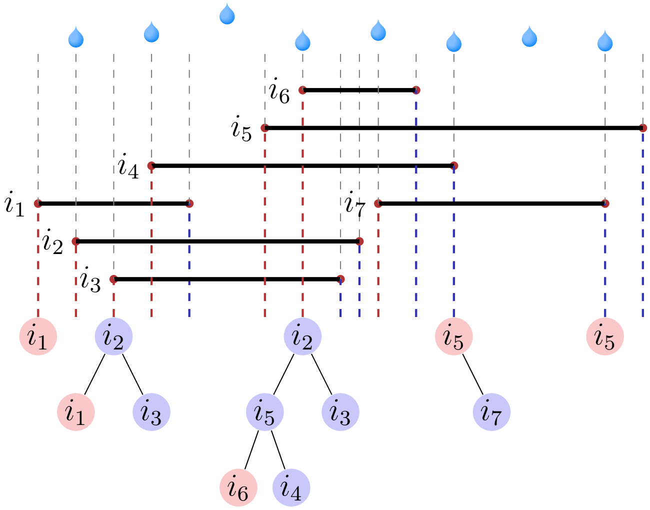


Q: How many intervals overlap maximally?

Sweep line controls a counter that is incremented (decremented) at the left (right) end point of an interval.

A: maximum counter value

Preparation: Overlapping Intervals

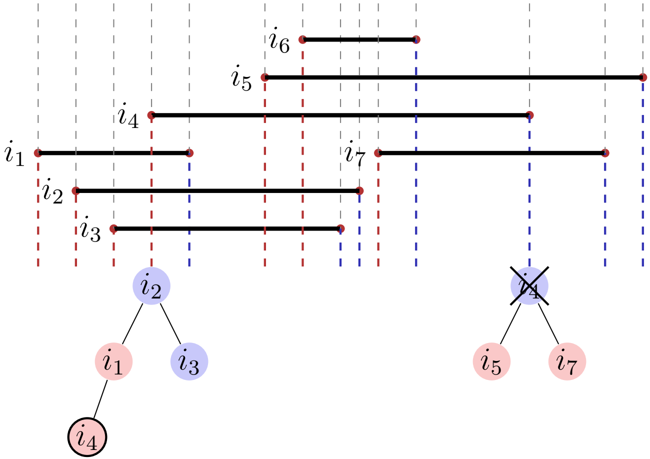


Q: Which intervals get wet?

Sweep line controls a binary search tree that comprises the intervals according to their vertical ordering.

A: intervalls on the very left of the tree.

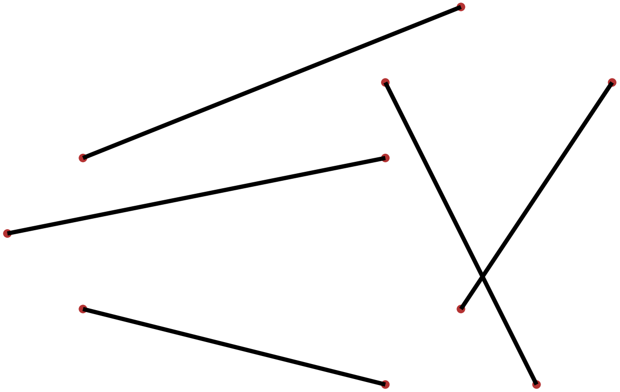
Preparation: Overlapping Intervals



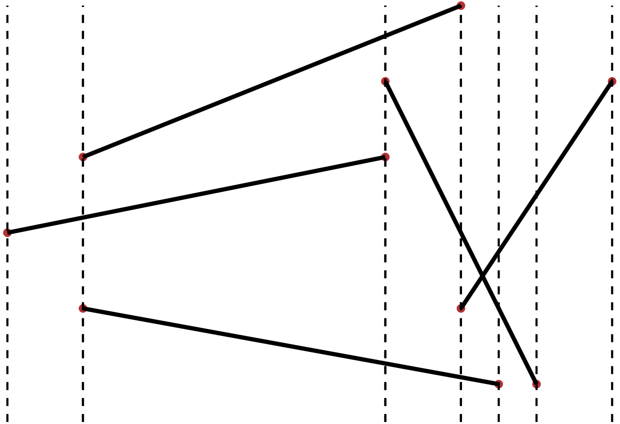
Q: Which intervals are neighbours?

A: intervalls on the very left of the tree.

Cutting many line segments



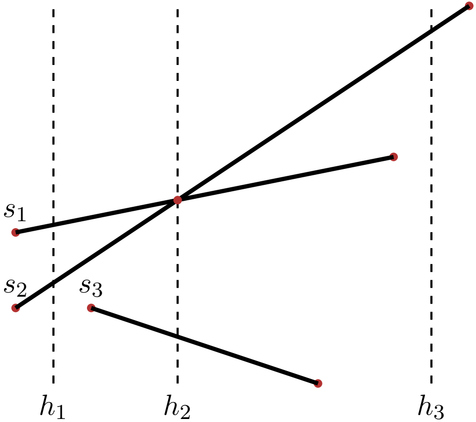
Sweepline Principle



Simplifying Assumptions

- No vertical line segments
- Each intersection is formed by at most two line segments.

(Vertical) Ordering line segments



Preorder (partial order without anti-symmetry)

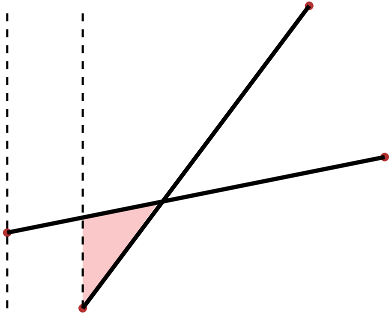
$$s_2 \preccurlyeq_{h_1} s_1$$
$$s_1 \preccurlyeq_{h_2} s_2$$
$$s_2 \preccurlyeq_{h_2} s_1$$
$$s_3 \preccurlyeq_{h_2} s_2$$

[37]

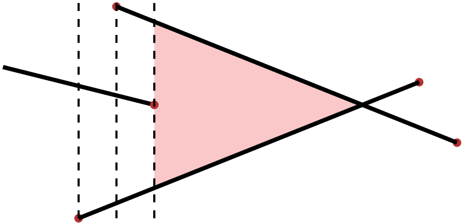
W.r.t. h_3 the line segments are uncomparable.

[37] No anti-symmetry: $s \preccurlyeq t \wedge t \preccurlyeq s \not\Rightarrow s = t$

Observation: two cases

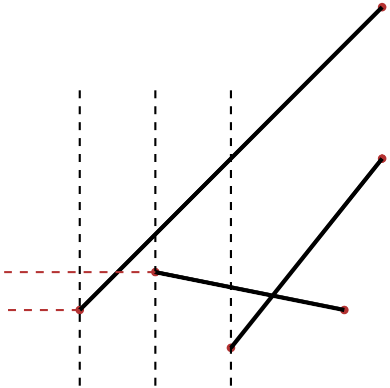


(a) Intersecting line segments are neighbours w.r.t. quasi-order from above directly from the start.



(b) Intersecting line segments are neighbours w.r.t. quasi-order from above after the last segment between them ends.

Observation: possible misunderstanding



It does not suffice to compare the y-coordinates of starting points of lines.
Positions on the sweep line have to be compared.

Moving the sweepline

- **Sweep-Line Status** : Relationship of all objects intersected by sweep-line
- **Event List** : Series of event positions, sorted by x-coordinate. Sweep-line travels from left to right and stops at each event position.

Sweep-Line Status

Preorder T of the intersected line segments

Required operations:

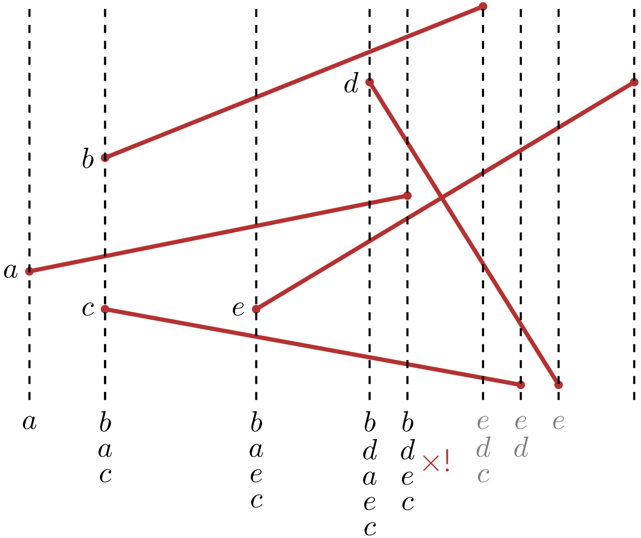
- **Insert(T, s)** Insert line segment s in T
- **Delete(T, s)** Remove s from T
- **Above(T, s)** Return line segment immediately above of s in T
- **Below(T, s)** Return line segment immediately below of s in T

Possible Implementation: Blanced tree (AVL-Tree, Red-Black Tree etc.)

Algorithm Any-Segments-Intersect(S)

Input: List of n line segments S
Output: Returns if S contains intersecting segments
$T \leftarrow \emptyset$
Sort endpoints of line segments in S from left to right (left before right and lower before upper)
for Sorted end points p **do**
    if p left end point of a segment s **then**
        Insert(T, s)
        if Above(T, s) $\cap s \neq \emptyset \ \vee$ Below(T, s) $\cap s \neq \emptyset$ **then** **return** true
    if p right end point of a segment s **then**
        if Above(T, s) $\cap$ Below(T, s) $\neq \emptyset$ **then** **return** true
        Delete(T, s)
return false;

Illustration



Analysis

Runtime of the algorithm Any-Segments-Intersect

- Sorting $\mathcal{O}(n \log n)$
- $2n$ iterations of the for loop. Each operation on the balanced tree $\mathcal{O}(\log n)$

Overal $\mathcal{O}(n \log n)$

24.4 Closest Point Pair

Closest Point Pair

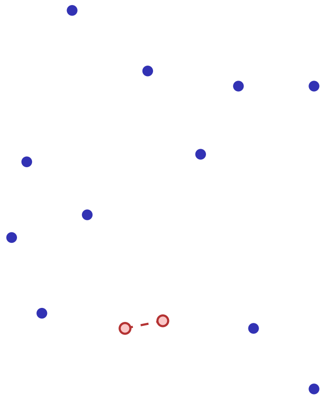
Euclidean Distance $d(s,t)$ of two points s and t:

$$d(s,t) = \|s-t\|_2 = \sqrt{(s_x - t_x)^2 + (s_y - t_y)^2}$$

Problem: Find points p and q from Q for which

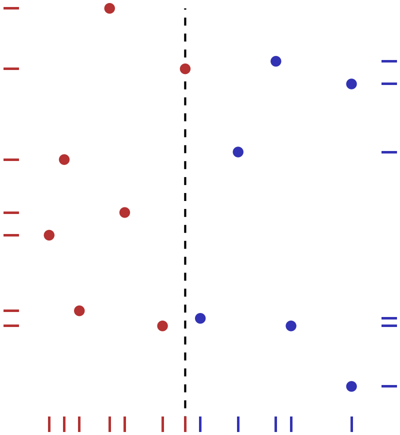
$$d(p,q) \le d(s,t) \ \forall\ s,t \in Q, s \neq t.$$

Naive: all $\binom{n}{2} = \Theta(n^2)$ point pairs.



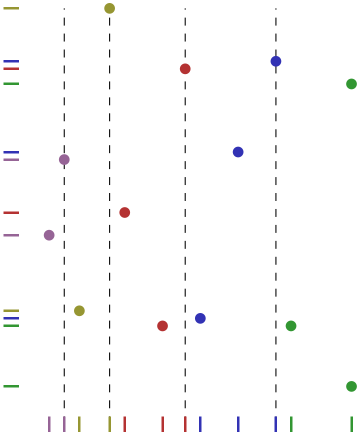
Divide And Conquer

- Set of points P, starting with $P \leftarrow Q$
- Arrays X and Y, containing the elements of P , sorted by x- and y-coordinate, respectively.
- Partition point set into two (approximately) equally sized sets P_L and P_R, separated by a vertical line through a point of P.
- Split arrays X and Y accordingly in X_L, X_R. Y_L and Y_R.



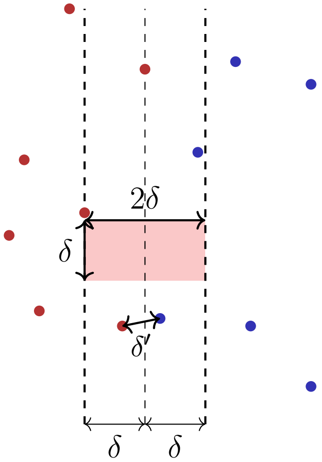
Divide And **Conquer**

- Recursive call with P_L, X_L, Y_L and P_R, X_R, Y_R. Yields minimal distances δ_L, δ_R.
- (If only $k \leq 3$ points: compute the minimal distance directly)
- After recursive call $\delta = \min(\delta_L, \delta_R)$. Combine (next slides) and return best result.

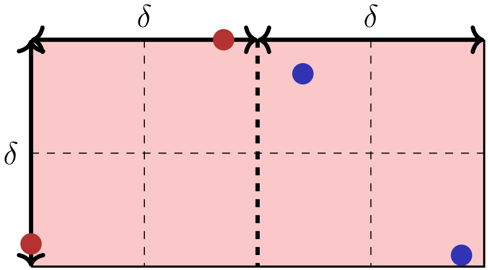


Combine

- Generate an array Y' holding y-sorted points from Y, that are located within a 2δ band around the dividing line
- Consider for each point $p \in Y'$ the maximally seven points after p with y-coordinate distance less than δ. Compute minimal distance δ'.
- If $\delta' < \delta$, then a closer pair in P than in P_L and P_R found. Return minimal distance.



Maximum number of points in the 2δ-rectangle



Two points in the $\delta/2 \times \delta/2$-rectangle have maximum distance $\frac{\sqrt{2}}{2}\delta < \delta$.
$\Rightarrow$ Square with side length $\delta/2$ contains a maximum of one point.
Eight non-overlapping $\delta/2 \times \delta/2$-Rectangles span the $2\delta \times \delta$ rectangle.

Implementation

- Goal: recursion equation (runtime) $T(n) = 2 \cdot T(\frac{n}{2}) + \mathcal{O}(n)$.
- Consequence: forbidden to sort in each steps of the recursion.
- Non-trivial: only arrays Y and Y'
- Idea: merge reversed: run through Y that is presorted by y-coordinate. For each element follow the selection criterion of the x-coordinate and append the element either to Y_L or Y_R. Same procedure for Y'. Runtime $\mathcal{O}(|Y|)$.

Overal runtime: $\mathcal{O}(n \log n)$.